**Pseudocode for ABCU Advising Program**

Menu:

1. Load the file data into the data structure

2. Print an alphanumerically ordered list of all courses

3. Print course title and prerequisites for a specific course

9. Exit

Function main():

Declare dataStructure

While True:

Print("Menu Options:")

Print("1. Load data")

Print("2. Print sorted list of courses")

Print("3. Print course details and prerequisites")

Print("9. Exit")

Input choice

If choice == 1:

Call loadData(dataStructure)

Else if choice == 2:

Call printSortedCourses(dataStructure)

Else if choice == 3:

Input courseID

Call printCourseDetails(dataStructure, courseID)

Else if choice == 9:

Print("Exiting program...")

Break

Else:

Print("Invalid option. Please try again.")

**Vector Data Structure**

Function loadData(vector):

Open file

For each line in file:

Parse course data (courseID, title, prerequisites)

Create courseObject(courseID, title, prerequisites)

Append courseObject to vector

Close file

Function printSortedCourses(vector):

Sort vector by courseID (O(n log n))

For each course in vector:

Print courseID

Function printCourseDetails(vector, courseID):

For each course in vector:

If course.courseID == courseID:

Print course.title

Print course.prerequisites

Return

Print("Course not found.")

**Hash Table Data Structure**

Function loadData(hashTable):

Open file

For each line in file:

Parse course data (courseID, title, prerequisites)

Create courseObject(courseID, title, prerequisites)

Insert courseID as key and courseObject as value into hashTable

Close file

Function printSortedCourses(hashTable):

Extract keys from hashTable

Sort keys alphanumerically (O(n log n))

For each key in sortedKeys:

Print key

Function printCourseDetails(hashTable, courseID):

If courseID exists in hashTable:

course = hashTable[courseID]

Print course.title

Print course.prerequisites

Else:

Print("Course not found.")

**Tree Data Structure**

Function loadData(tree):

Open file

For each line in file:

Parse course data (courseID, title, prerequisites)

Create courseObject(courseID, title, prerequisites)

Insert courseObject into tree by courseID

Close file

Function printSortedCourses(tree):

Perform in-order traversal of tree

For each node in traversal:

Print node.courseID

Function printCourseDetails(tree, courseID):

node = tree.search(courseID)

If node is not null:

Print node.title

Print node.prerequisites

Else:

Print("Course not found.")

**Runtime Analysis**

Vector:

- Load: O(n) (append operations)

- Sort: O(n log n)

- Search for a course: O(n) (linear search)

Hash Table:

- Load: O(n) (hashing insertions)

- Sort: O(n log n) (sorting extracted keys)

- Search for a course: O(1) (average case)

Tree:

- Load: O(n log n) (insertions into binary search tree)

- Sort: O(n) (in-order traversal)

- Search for a course: O(log n) (binary search)

**Evaluation of Data Structures**

Vector:

- Advantages: Simple to implement, efficient for small datasets.

- Disadvantages: Sorting and searching are slower for large datasets.

Hash Table:

- Advantages: Fast lookups, suitable for large datasets

- Disadvantages: Requires additional sorting for alphanumeric order.

Tree:

- Advantages: Maintains sorted order, efficient search and retrieval.

- Disadvantages: Insertions and deletions are slower compared to hash tables.

**Recommendation/Plan**

I plan to use the Tree Data Structure because:

- It ensures the courses remain sorted for easy retrieval (in-order traversal).

- Provides efficient search (O(log n)) and sorted operations without additional overhead like in hash tables.